**Hangul Recognition with Keras**

This is the documentation which goes along with the code in the GitHub repository

PatrickHuynh97/HangulRecognition

For more information on character recognition (MNIST) and how a Convolutional Neural Network works, please the document in another one of my repositories:

PatrickHuynh97/CNNMNISTrecognition

This project is very similar to that in the repository below, except I am using Keras instead of Estimators. In some sections I may use code from this repository to assist in this project.

IBM/tensorflow-hangul-recognition

Written by Patrick Huynh

July 2018

Table of Contents

[Keras 4](#_Toc518649386)

[CNN Structure 4](#_Toc518649387)

# Keras

Keras is a high-level neural network API in TensorFlow. It makes it extremely easy to create prototype neural networks without having to deal with the nitty gritty stuff that happens in the background. For this project, we will define a Convolutional Neural Network to classify handwritten Hangul characters, and then deploy this on an Android application for inference. The end goal is that this trained model can be used to help people learn the language in a more practical way, using their finger to draw the characters and promote muscle memory.

# CNN Structure

The CNN we use here will have the same structure as the network defined in *IBM/tensorflow-hangul-recognition*. The structure is as follows:

* **Convolutional layer**
  + kernel size = [5, 5]
  + feature maps = 32
  + strides = 1
  + activation function = ReLU
  + padding = SAME
* **Max Pooling**
  + filter size = [2, 2]
  + stride = 2
  + padding = SAME
* **Convolutional Layer**
  + kernel size = [5, 5]
  + feature maps = 64
  + strides = 1
  + activation function = ReLU
  + padding = SAME
* **Max Pooling**
  + filter size = [2, 2]
  + stride = 2
  + padding = SAME
* **Convolutional Layer**
  + kernel size = [3, 3]
  + feature maps = 128
  + strides = 1
  + activation function = ReLU
  + padding = SAME
* **Max Pooling**
  + filter size = [2, 2]
  + stride = 2
  + padding = SAME

This network differs slightly from that which was defined on the TensorFlow Layers tutorial. An extra convolutional layer has been added with 128 feature maps, to help account for the very large number of possible output classes. We will see later how one defines a CNN like this in Keras.

# Input data

Before we create our model, we need to find a large amount of data that can be used to train it. Luckily, a project like this has been done before, so the code to convert *.ttf* files into JPEG already exists. We download fonts from the website **naver**, and then use a python script to convert these images into JPEG format.

Keras provides a function to convert JPEG images into a form it can read, making the process of creating an input pipeline very easy. We can also perform some image augmentations in Keras to further expand our training set.